JOHNSON GRANT
IN-60-CR
20009
P-8

T on the Cray X/MP

Final Report

(NASA-CR-188590) T ON THE CRAY X/MP Final Report (Houston Univ.) 8 p CSCL 09B N91-25622

Unclas
G3/60 0020009

Paul Hudak
Yale University

April, 1991

Cooperative Agreement NCC 9-16
Research Activity No. AI.07

NASA Johnson Space Center
Information Systems Directorate
Information Technology Division

Research Institute for Computing and Information Systems
University of Houston - Clear Lake

T·E·C·H·N·I·C·A·L R·E·P·O·R·T

The RICIS Concept

The University of Houston-Clear Lake established the Research Institute for Computing and Information systems in 1986 to encourage NASA Johnson Space Center and local industry to actively support research in the computing and information sciences. As part of this endeavor, UH-Clear Lake proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a three-year cooperative agreement with UH-Clear Lake beginning in May, 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The mission of RICIS is to conduct, coordinate and disseminate research on computing and information systems among researchers, sponsors and users from UH-Clear Lake, NASA/JSC, and other research organizations. Within UH-Clear Lake, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business, Education, Human Sciences and Humanities, and Natural and Applied Sciences.

Other research organizations are involved via the "gateway" concept. UH-Clear Lake establishes relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research.

A major role of RICIS is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. Working jointly with NASA/JSC, RICIS advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research, and integrates technical results into the cooperative goals of UH-Clear Lake and NASA/JSC.

T on the Cray X/MP

Final Report

Preface

This research was conducted under auspices of the Research Institute for Computing and Information Systems by Dr. Paul Hudak, Department of Computer Science, Yale University. Dr. Terry Feagin served as RICIS research coordinator.

Funding has been provided by Information Technology Division, Information Systems Directorate, NASA/JSC through Cooperative Agreement NCC 9-16 between NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA technical monitor for this activity was Robert T. Savely, of the Software Technology Branch, Information Technology Division, Information Systems Directorate, NASA/JSC.

T on the Cray X/MP
– Final Report –

Principal Investigator: Prof. Paul Hudak
Department of Computer Science
Yale University
New Haven, CT

April 1991

1 Background

This is the final report on research conducted under the NASA Johnson Space Center and administered through the RICIS Project at the University of Houston–Clear-Lake.

The T programming language [4, 5] is a superset of Scheme [1], which is in turn a dialect of Lisp. The proposed research was aimed at porting T to the Cray X/MP computer. The effort involved research into unique systems engineering and software engineering problems related not only to compiler design and implementation, but also to parallel computation, the X/MP being a shared-memory multiprocessor.

2 Research Objectives

In 1986 the T Project at Yale completed construction of a optimizing compiler for T called *Orbit* [2] which is incorporated into a flexible run-time environment called the *T System*. T programs compiled using Orbit have been benchmarked against similar programs in conventional languages such as C and Pascal, with comparable results. The T System is available on most MC68000-based workstations (Sun, Apollo, HP), as well as Sun SparcStations, DecStations, and MIPS workstations. In addition, a parallel dialect called *Mul-T* [3] is available on the Encore Multimax.

Given our excellent technology base on workstations, our primary goal in this research was to investigate the feasibility of porting our technology to the Cray X/MP. Doing so would involve the sub-tasks of retargeting the assembler, code-generator, and run-time system.

3 Approach

As an intermediate language, Orbit uses a restricted version of the lambda calculus called CPS (for continuation-passing style). Procedures do not return but rather take as an argument a procedure to call upon completion, passing to it any results. One of the benefits of CPS is that control flow becomes more explicit and is more easily manipulated by the optimizer. The interfaces between T and the CPS intermediate language, and between the intermediate language and the code generator, are simple and well-defined. This makes it relatively easy to adapt the compiler to changes in T and to retarget the compiler to different machines.

Most optimizations are done via source-to-source transformations of the CPS intermediate language. The expressiveness, simplicity, and well understood semantics of CPS make optimizations easy to implement, and with confidence that the optimized code will be correct. The optimizations already implemented include eliminating tail recursive calls, flushing unused variables and arguments, evaluating tests for effect, substituting variables so as to eliminate unnecessary calls, integrating procedures, etc.

Moving a compiler from one machine to another is traditionally a difficult task, given the great differences between most machines at the architectural level. Some researchers have made attempts to write "table-driven code-generators," but this usually results in a compromise in efficiency, since such systems make it difficult to either take advantage of, or avoid problems with, a particular machine's idiosyncrasies. Our approach with Orbit has been a compromise between a fully table-driven system and a more customized approach. The result is a reasonably effective porting methodology that is relatively straightforward, although more difficult than table-driven approaches.

As is true of any large software system, there are also certain machine-dependent features in the runtime system, which we capture in the T System's "runtime library." This includes the storage manager, the file system interface, the exception handling subsystem, and the local operating system interface. These modules will require modification for the X/MP.

4 Results

Although we eventually managed to re-target the assembler and code generator (the most critical components of the overall plan), it was much more difficult than anticipated. There were several reasons for this, none of which were under our control:

1. Our access to a Cray X/MP was via remote access at the Pittsburgh Supercomputer Center. Unfortunately, the network connection was slow and unreliable, making software development quite difficult.

2. The program development/debugging facilities available on the Cray X/MP were far more primitive than what we were accustomed to on our "more sophisticated" workstations. This also slowed down our software development activity considerably.

3. Perhaps most important of all, our entire proposal was predicated on the availability of Unix on the Cray X/MP (Unicos), something that at the time we were writing the proposal was promised to be available by the time the grant was awarded. In fact Unix was not made available until half way through the contract period, and even then the initial releases were buggy and generally unreliable. This was a very serious impediment to our progress.

Because of these difficulties, we were not able to complete the entire port of T to the Cray X/MP. In particular, the run-time system was not ported, and thus T programs cannot be run without explicit linking of the required run-time support.

Aside from the above problems, we found that the run-time performance gain we achieved in generated code was quite disappointing. Upon a thorough (but difficult, again because of the poor program development facilities), we discovered that as a *scalar* machine the Cray X/MP was only about 5 times faster than a MC68020 microprocessor, and that was indeed the kind of speed-ups that we achieved. In order to achieve any higher performance, we have determined that the Cray X/MP would have to be exploited in two non-trivial ways:

1. The pipeline nature of the hardware would have to be exploited by suitable code generation techniques that minimized jumps.

2. The vector and floating-point hardware would have to be exploited to ensure fast numeric performance.

Unfortunately, both of these improvements represent major redesigns of our compiler and code generator, and were outside the scope of this contract. In addition, it is not entirely clear, without further research, whether these improvements will actually pay off, since:

1. Typical Lisp programs (representing, for example, AI program development) involve very unpredictable branching behavior, due to the dynamic nature of the applications. This makes generating large sections of jump-free code quite difficult, thus reducing the chances of exploiting the pipelined architecture.

2. Similarly, most Lisp applications are not intensive in numeric and vector computations, thus reducing the chances of exploiting the vector/floating-point hardware.

Although these results are somewhat negative, they are useful discoveries, and should aid in future development of Lisp systems on Cray X/MP or similar supercomputer architectures.

References

[1] Clinger, W. et al. The revised revised report on scheme, or an uncommon lisp. AI Memo 848, Massachusetts Institute of Technology, August 1985.

[2] D. Kranz, R. Kelsey, J. Rees, P. Hudak, J. Philbin, and N. Adams. Orbit: an optimizing compiler for Scheme. In *SIGPLAN '86 Symposium on Compiler Construction*, pages 219–233. ACM, June 1986. Published as SIGPLAN Notices Vol. 21, No. 7, July 1986.

[3] D.A. Kranz, R.H. Halstead, and E. Mohr. Mul-T: A high-performance parallel Lisp. In *Proceedings of 1989 SIGPLAN Conference on Programming Language Design and Implementation*, pages 81–90. ACM/SIGPLAN, June 1989.

[4] J.A. Rees and N.I. Adams. T: a dialect of lisp or, lambda: the ultimate software tool. In *Proceedings 1982 ACM Conference on LISP and Functional Programming*, pages 114–122. ACM, August 1982.

[5] J.A. Rees, N.I. Adams, and J.R. Meehan. The t manual. Technical Report 4th edition, Yale University, January 1984.